THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
EUROPE

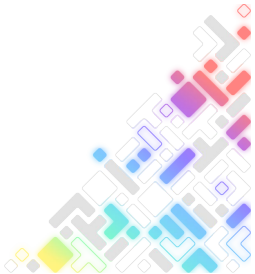
The Case for an SoC Power Management Driver

September 17, 2024
Vienna, Austria

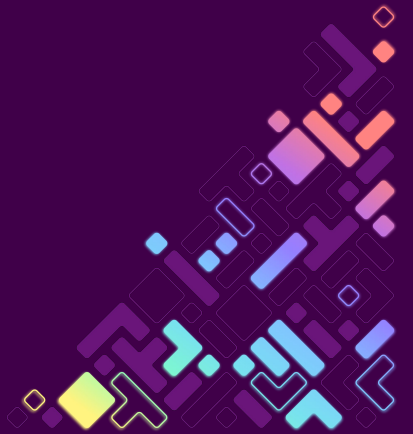
THE LINUX FOUNDATION #ossummit s{w}boyd@{kernel,chromium}.org

Agenda

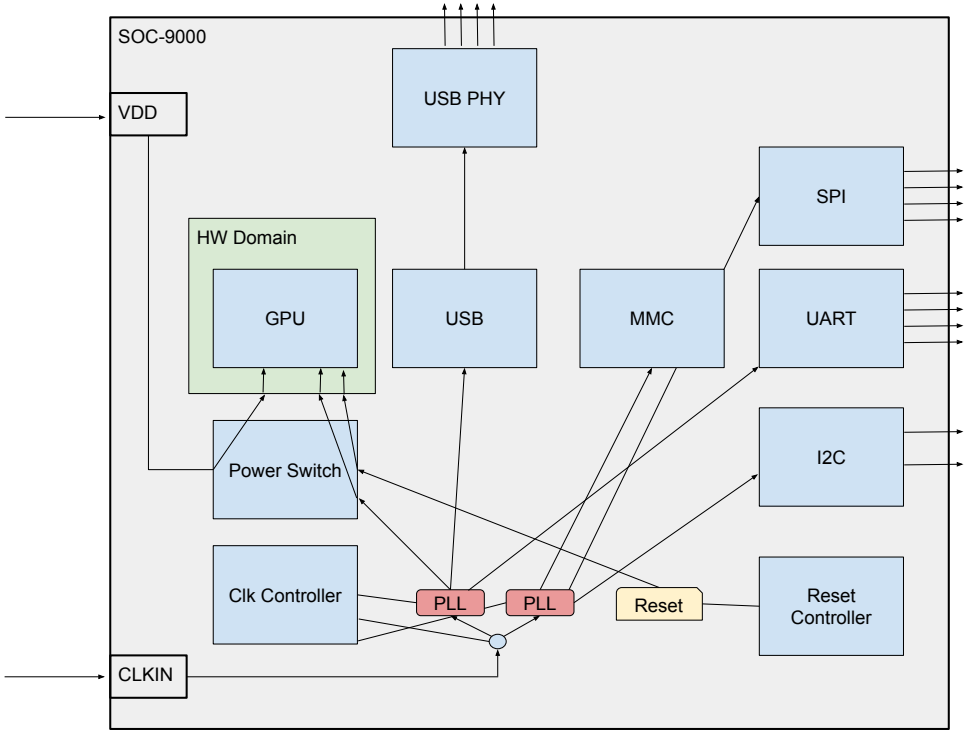
- Background
- Power management approaches
- Problem
- Previous and ongoing attempts
- Proposed solution
- Questions



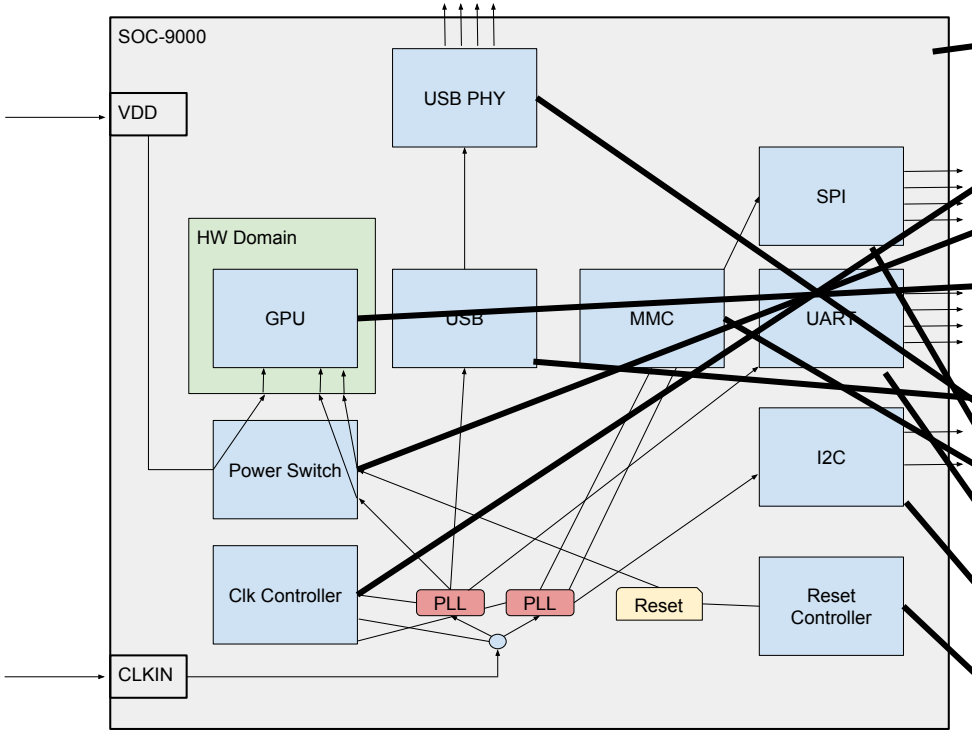
What’s an SoC?



System-on-Chip (SoC)

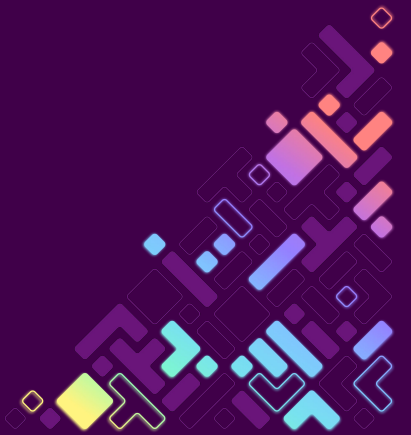


DT and SoCs



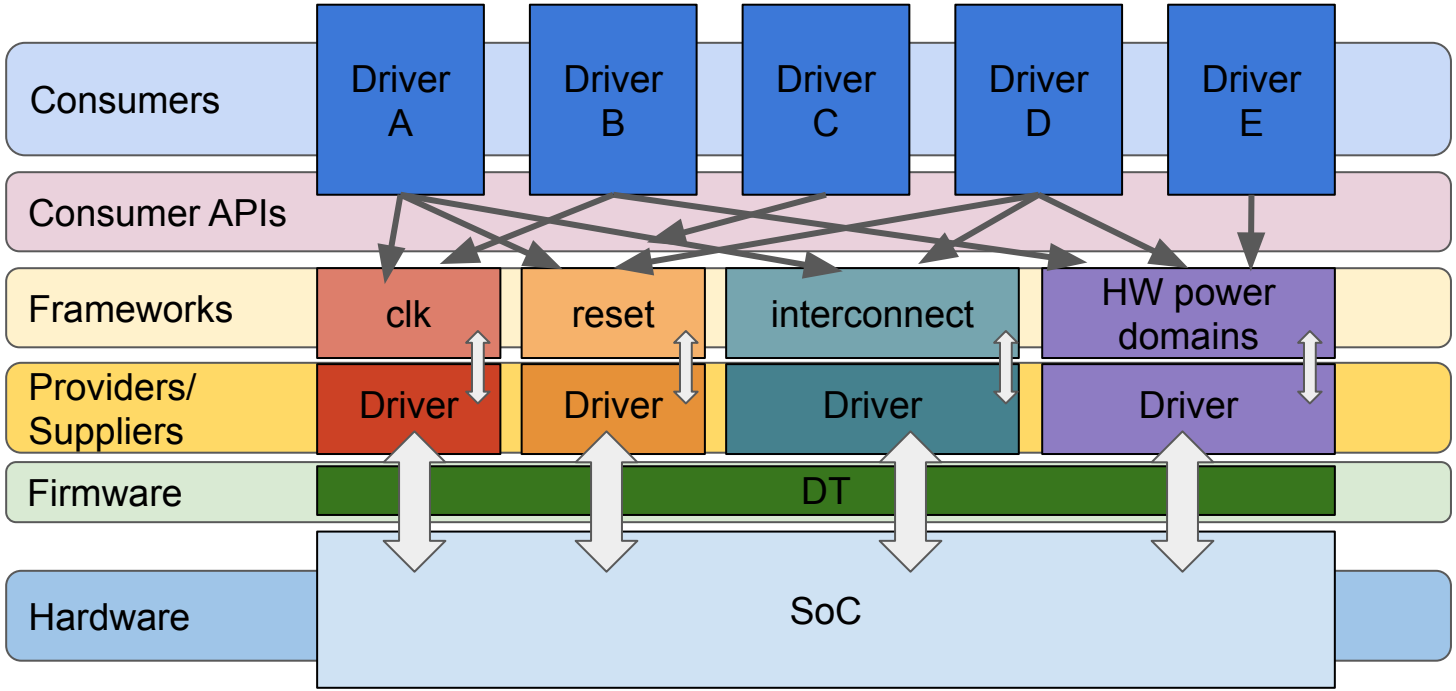

```
soc@0 {
  compatible = “simple-bus”;
  clk: clock-controller { };
  pwr: power-controller { };
  gpu { clocks = <&clk 0>;
     resets = <&rst 0>;
     power-domains = <&pwr> };
  usb { clocks = <&clk 1>; };
  phy: phy { };
  mmc { clocks = <&clk 2>; };
  spi { clocks = <&clk 3>; };
  uart { clocks = <&clk 4>; };
  i2c { clocks = <&clk 5>;};
  rst: reset-controller { };
};
```

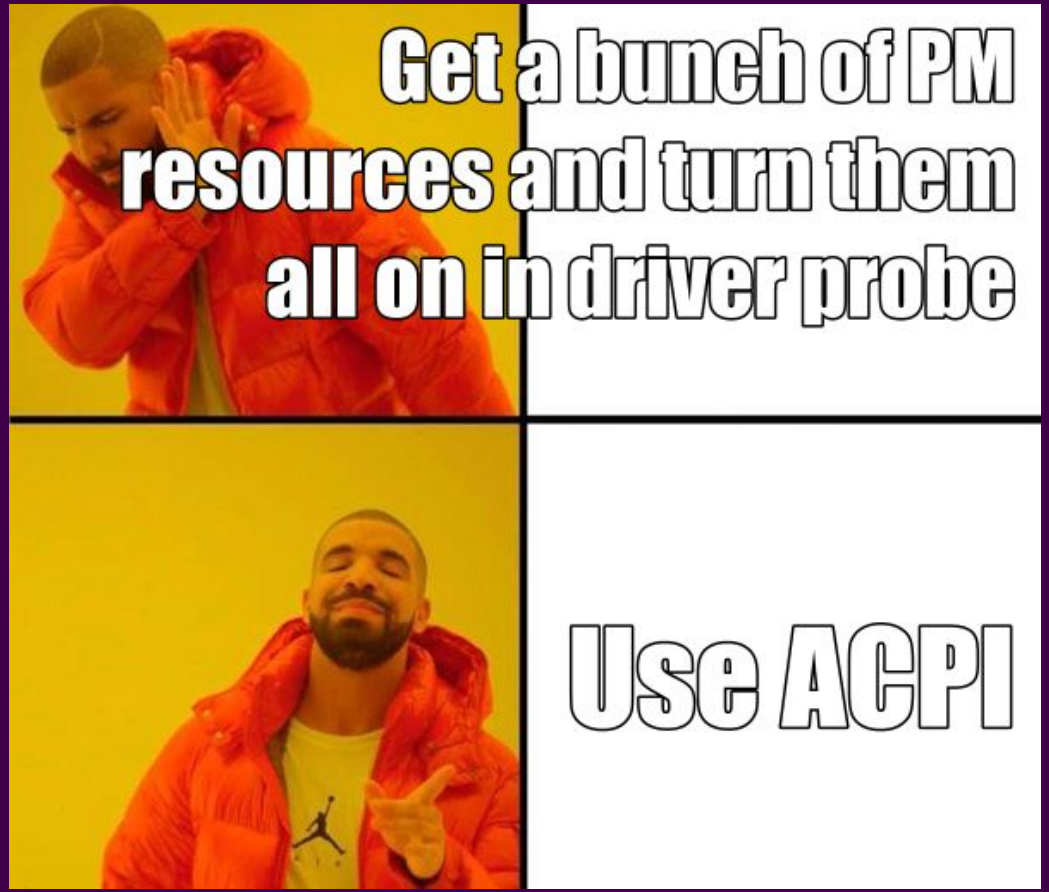
Power Management Approaches



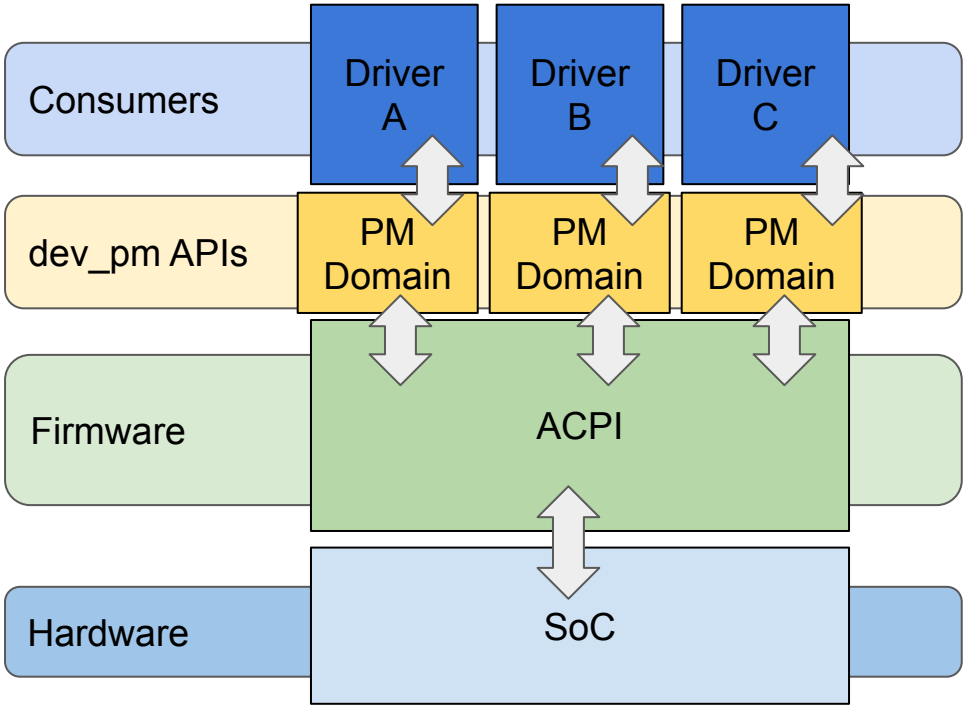

Implementing PM w/ DT
This little maneuver is gonna cost
a dozen kernel frameworks

Device Power Management w/ DT




Get a bunch of PM resources and turn them all on in driver probe
Use ACPI

Device Power Management w/ ACPI



Device PM Domains

- dev_pm_ops is the “one” PM framework
- Coordinates PM transitions for device
- Only one PM domain per ‘struct device’

```
struct device {

        struct dev_pm_domain *pm_domain;

        …

};
```

file: include/linux/pm.h

```
struct dev_pm_domain {
        struct dev_pm_ops ops;
        int (*start)(struct device *dev);
        void (*detach)(struct device *dev, bool power_off);
        int (*activate)(struct device *dev);
        void (*sync)(struct device *dev);
        void (*dismiss)(struct device *dev);
        int (*set_performance_state)(struct device *dev,
unsigned int state);
};
```

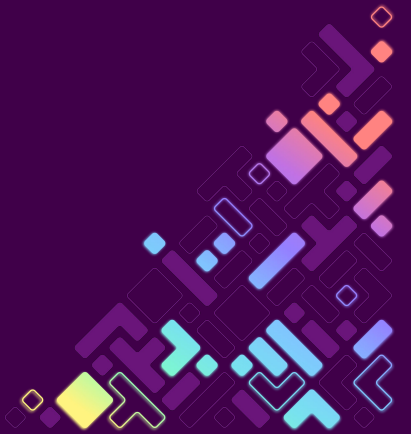
Device Power Management w/ ACPI

- ACPI spec defines device power states [6]
  - D0 (on)
  - D3 (off)
- Implement power states via PM domains
  - Fills out dev_pm_ops in PM domain
  - Boils down to acpi_device_set_power()
- Runtime and System PM decides device power state
- Driver probe puts device into D0
- Driver remove puts device into D3
- Subsystem frameworks rarely if ever used
  - ACPI firmware hides SoC PM implementation details

file: drivers/acpi/device_pm.c

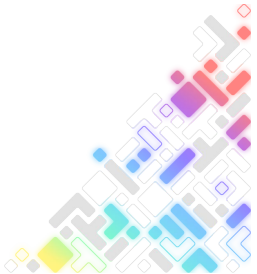
```
static struct dev_pm_domain acpi_general_pm_domain = {
        .ops = {
                .runtime_suspend = acpi_subsys_runtime_suspend,
                .runtime_resume = acpi_subsys_runtime_resume,
#ifdef CONFIG_PM_SLEEP
                .prepare = acpi_subsys_prepare,
                .complete = acpi_subsys_complete,
                .suspend = acpi_subsys_suspend,
                .resume = acpi_subsys_resume,
                .suspend_late = acpi_subsys_suspend_late,
                .suspend_noirq = acpi_subsys_suspend_noirq,
                .resume_noirq = acpi_subsys_resume_noirq,
                .resume_early = acpi_subsys_resume_early,
                .freeze = acpi_subsys_freeze,
                .poweroff = acpi_subsys_poweroff,
                .poweroff_late = acpi_subsys_poweroff_late,
                .poweroff_noirq = acpi_subsys_poweroff_noirq,
                .restore_early = acpi_subsys_restore_early,
#endif
        },
};
```

What’s the problem?



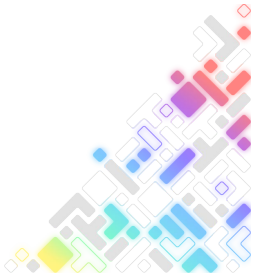
Problems In DT World

- Portable drivers must be firmware aware
- Device power state at probe unknown
- SoC PM integration details spread throughout driver tree
- PM resources are **also** provided by platform devices
- Overall SoC power state unknown
- Unable to coordinate power off of unused device PM resources

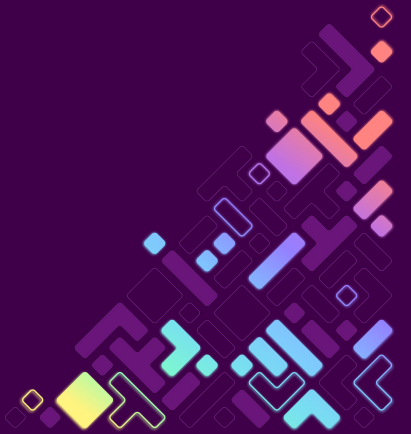


What Do We Want In a Solution!?

- Work with existing DTs
  - Backwards compatible (no flag days!)
- Remove SoC PM details from platform drivers
- Work with existing drivers that use PM resources directly
- Provide consistent device power states
- Power on/off device PM resources at probe/remove
- Bonus: Power off unused device PM resources
- Bonus: Extend usage of runtime PM



Previous && Ongoing Solutions



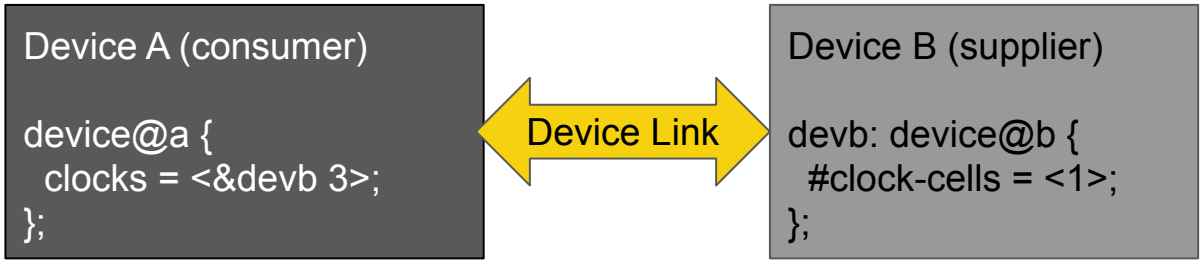
Solving Probe Order

- Resource/Tracking allocation framework ❌
- On-demand device probing ❌
- Device links ✅
- fw devlink ✅

✅ Merged

❌ Rejected

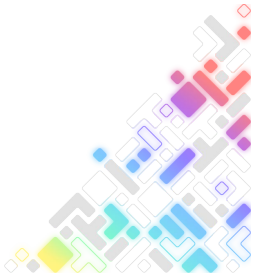
Theme: Probe defer loops caused by PM suppliers



Q: When Can We Turn Off Unused PM Resources?

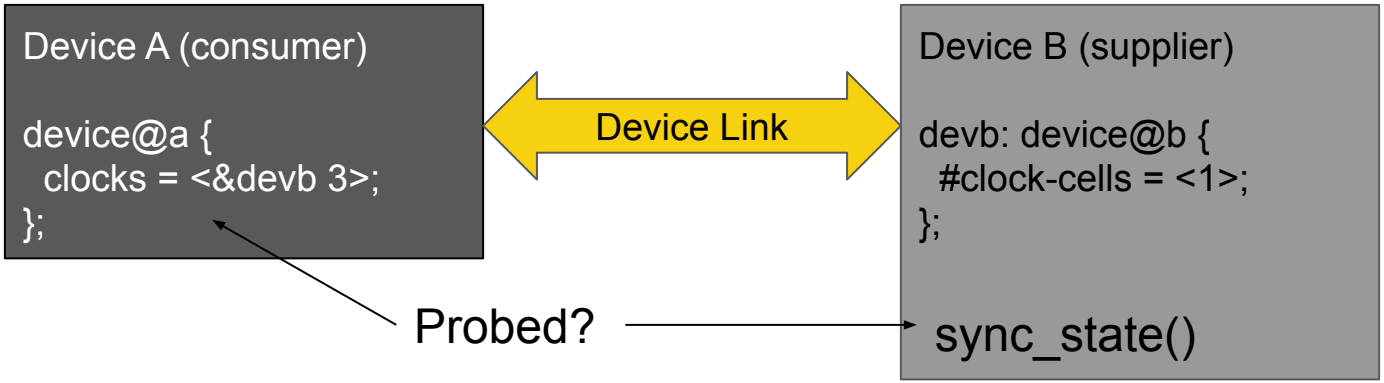
- A: When device PM resources are known to be unused
  - Right … that's recursive
- Q: How do we know they're unused?
  - A: When the consumer device is marked disabled in DT (status = "disabled")
  - A: When the consumer driver isn't enabled (# CONFIG_DRIVER is not set)
- Q: Great! How about when the driver is builtin or a module?
  - A: When all consumers of PM supplier have probed

And that's the story of how sync_state() was born



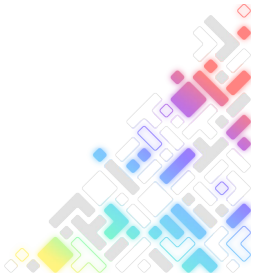
struct device_driver::sync_state()

- Part of fw devlink series
- Idea: If we've solved probe ordering we can power down unclaimed resources
- Calls sync_state() after all consumers of supplier device are probed
  - Relies on device links to know which consumers are still unbound



Problems with sync_state()

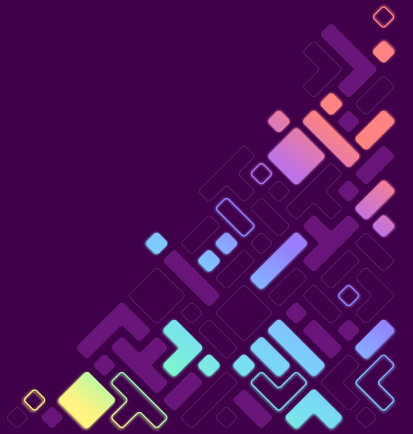
- Power off sequencing uncoordinated between kernel subsystems
- Assumes PM resources are provided by device drivers in the kernel
  - Hello ACPI
- Requires consumers drivers to probe
- fw devlink isn’t always 100% correct
  - Device links are an unreliable signal



ACPI

ACPI solves all these problems, why can’t we use it?

Brainstorm



PM Domains as Middleware

- Earlier solutions take a bottom-up or top-down approach
  - Resource tracker - Top-down
    - “Devices are consumers and consumers are in control”
  - Device links + sync_state() - Bottom-up
    - “There’s consumers and suppliers, and suppliers are in control”
- Middle-out approach
  - PM domains sit in the middle between suppliers and consumers
  - Coordinates power sequencing between kernel frameworks
  - PM resources are influenced by consumer device power state
  - Shared PM resources are managed by suppliers but influenced by consumers

Generic PM Domains

- “Inherits” dev_pm_domain
- Plumbed into driver core similar to ACPI
  - Powers on at probe
  - Powers off at remove
- Nestable
- May contain many devices
  - Coordinate power among devices
- DT bindings supported
  - #power-domain-cells
  - power-domains

file: include/linux/pm_domain.h

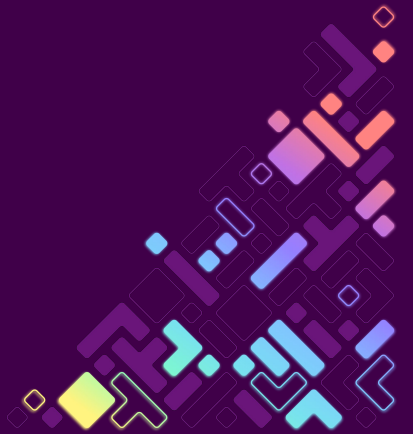
```
struct generic_pm_domain {
  struct dev_pm_domain domain;
  int (*power_off)(struct generic_pm_domain
                   *domain);
   int (*power_on)(struct generic_pm_domain
                   *domain);
  …
};
```


Generic PM domains
It's perfectly safe.

Barriers to Adoption of Generic PM Domains w/ DT

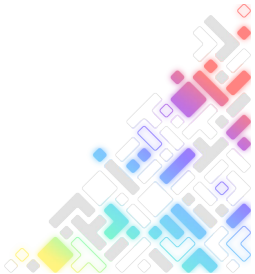
- power-domains DT property encourages **all** PM domains to be in DT
  - DT property mostly used for hardware power switches
  - Could fake it by adding #power-domain-cells to all PM supplier devices
  - Could fake it by adding #power-domain-cells to soc node and power-domains to child nodes
- of_platform_default_populate() creates and adds platform devices in one call
  - Prevents getting between driver probe to associate PM domain
- Only one PM domain per-device
  - Devices with power-domains DT property are already “claimed”
- Removing direct control of PM resources difficult sometimes
  - Some devices have specific clk frequency requirements that are calculated dynamically

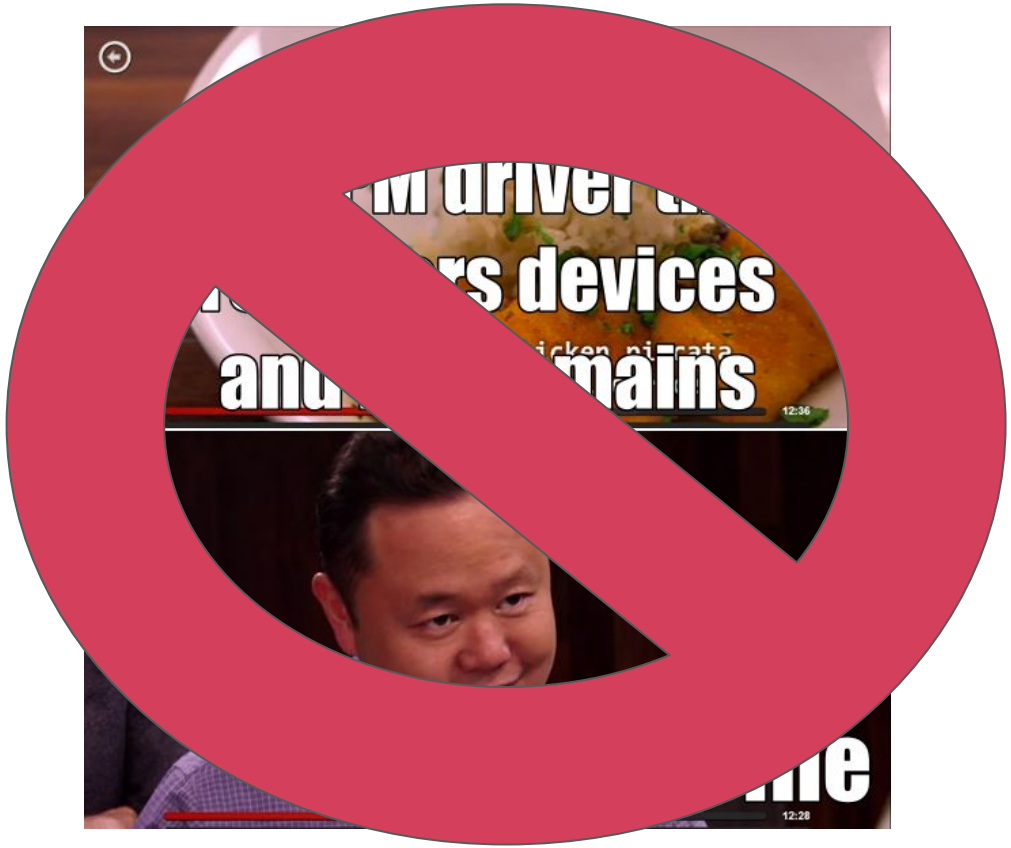
SoC PM Driver



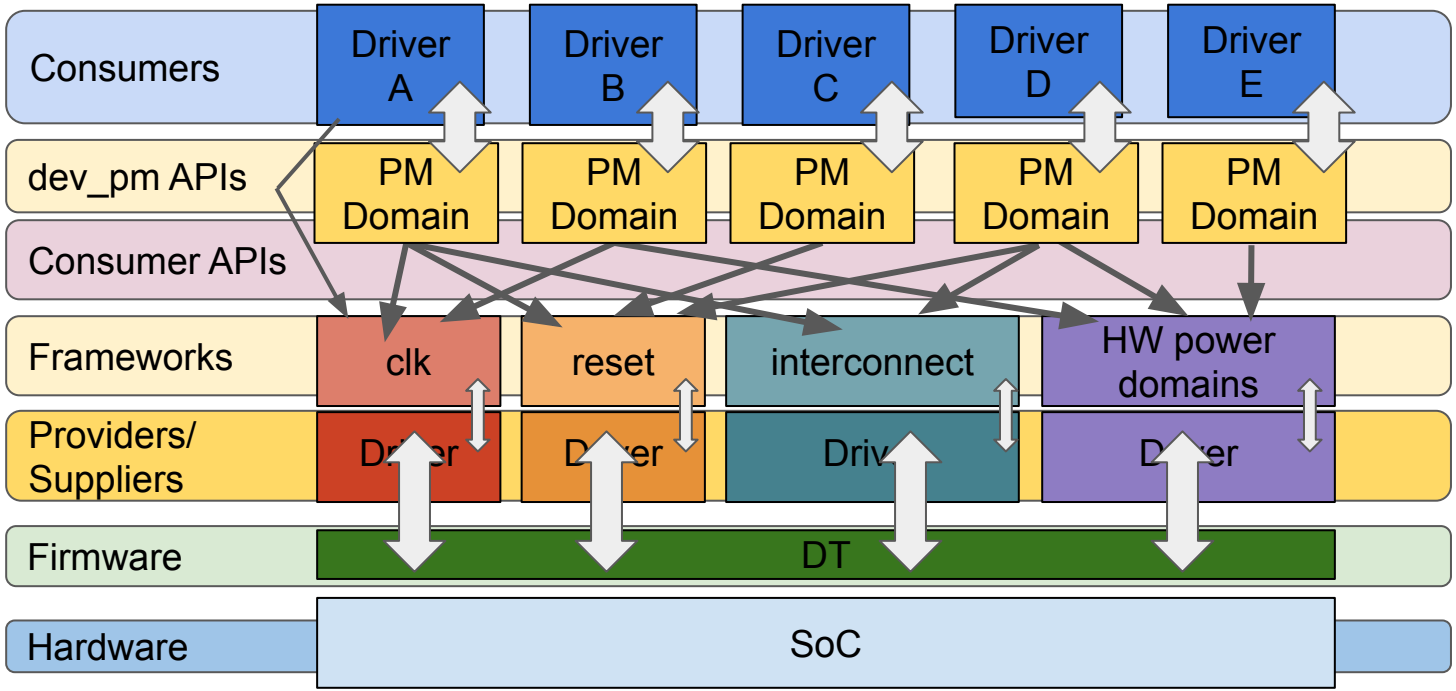
TL;DR

- Bind a driver to the SoC node’s device
- Associate PM domains with child platform devices
- Move PM code from platform drivers to PM domains
- Profit $$

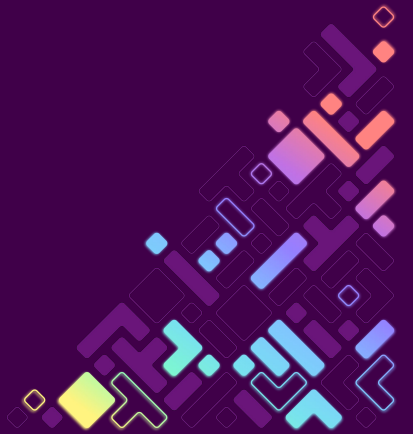



rs devices
mains
12:36
12:28

Block Diagram



Implementation

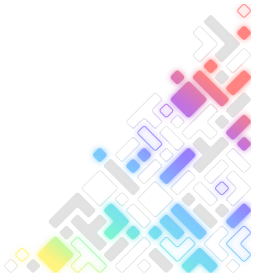


Hook Device Creation

- Introduce driver for soc node
  - Match compatible like “vendor,soc-9000”
- Populate child nodes as child platform devices
  - Call of_platform_default_populate()

```
struct platform_driver soc9000_driver = {
        .driver.probe = of_platform_default_populate(),
        .driver.match_table = { “vendor,soc-9000” },
};
```

Success! Hooked device creation for soc node’s children



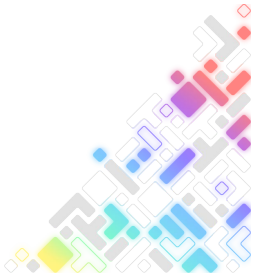
Add PM Domains

Need to associate PM domains with child devices before adding to platform bus

- Split of_platform_default_populate() in half
  - Creation of platform device
  - Addition of platform device to bus
- Add PM domain after creation and before addition

```
pdev = of_platform_device_alloc();
dev_pm_domain_set(&pdev->dev, pm_domain);
of_platform_device_add(pdev);
```

Success! Associated PM domain with device



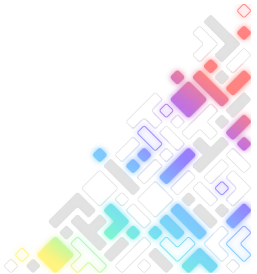
Get PM Resources in PM Domain

Use struct dev_pm_domain::activate()

```
int get_pm_resources(struct device *dev)
{
        clk_get(dev, …);
        regulator_get(dev, …);
        …
}
pm_domain.activate = get_pm_resources;
```

Success! PM domain gets resources before device is bound

Note: We need to wrap this for struct generic_pm_domain

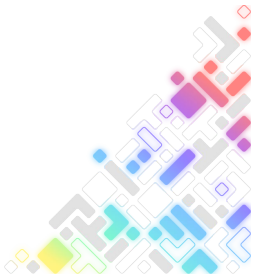


Manage PM Resources in PM Domain

Use struct generic_pm_domain::power_{on,off}()

```
int power_on_pm_resources(struct device *dev)
{
        clk_prepare_enable(clk);
        regulator_enable(regulator);
        …
}
generic_pm_domain.power_on = power_on_pm_resources;
generic_pm_domain.power_off = power_off_pm_resources;
```

Success! PM domain powers on and off PM resources for device

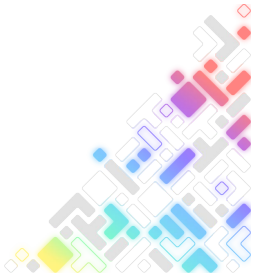


Remove PM Code

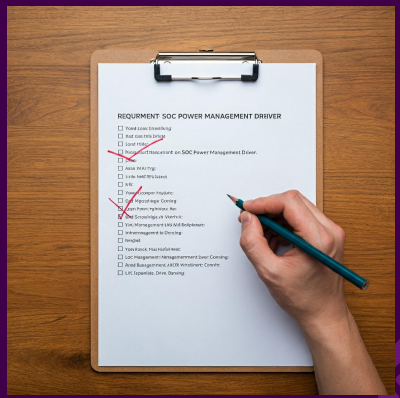
- Set struct device::platform_data to something non-NULL
- Check dev_get_platdata() for non-NULL in driver
  - Skip getting PM resources and code that isn't used
  - Remove PM code after all SoCs converted

```
dev.platform_data = (void *)1UL;
if (dev_get_platdata(dev))
        skip_pm_stuff();
```

Success! Platform drivers don't have SoC PM details



Requirements Check



Work With Existing DTs ✅

- ✓ **Work with existing DTs**
- ❑ Remove SoC PM details from platform drivers
- ❑ Work with existing drivers that use PM resources directly
- ❑ Provide consistent device power states
- ❑ Power on device PM resources at probe
- ❑ Power off device PM resources at remove
- ❑ Bonus: Power off unused device PM resources
- ❑ Bonus: Extend usage of runtime PM

```
soc@0 {
  compatible = “soc-9000”,
                “simple-bus”;
  device@ffad9000 {
    ....
  };
};
```

Remove SoC PM Details From Drivers ✅

- ✓ Work with existing DTs
- ✓ **Remove SoC PM details from platform drivers**
- ❑ Work with existing drivers that use PM resources directly
- ❑ Provide consistent device power states
- ❑ Power on device PM resources at probe
- ❑ Power off device PM resources at remove
- ❑ Bonus: Power off unused device PM resources
- ❑ Bonus: Extend usage of runtime PM

file: include/linux/pm.h

```
struct dev_pm_domain {
  void (*detach)(struct device *dev, bool
power_off);
  int (*activate)(struct device *dev);
  void (*dismiss)(struct device *dev);
};

struct generic_pm_domain {
  struct dev_pm_domain domain;
  int (*power_off)(struct generic_pm_domain*);
  int (*power_on)(struct generic_pm_domain*);
};
```

Work With Existing Drivers ✅

- ✓ Work with existing DTs
- ✓ Remove SoC PM details from platform drivers
- ✓ **Work with existing drivers that use PM resources directly**
- ❑ Provide consistent device power states
- ❑ Power on device PM resources at probe
- ❑ Power off device PM resources at remove
- ❑ Bonus: Power off unused device PM resources
- ❑ Bonus: Extend usage of runtime PM

DT binding doesn't need to change

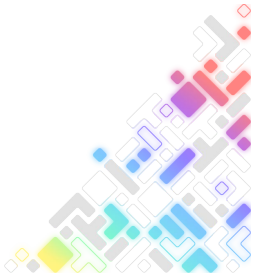
```
device@ffad9000 {
    clocks = <&sup DEV_CLK>;
    vdd-supply = <&pp1800>;
    interconnects = <&bus0>;
    power-domains = <&pmdomain>;
};
```

Consumer APIs still work

Provide Consistent Device Power States ✅

- ✓ Work with existing DTs
- ✓ Remove SoC PM details from platform drivers
- ✓ Work with existing drivers that use PM resources directly
- ✓ **Provide consistent device power states**
- ❑ Power on device PM resources at probe
- ❑ Power off device PM resources at remove
- ❑ Bonus: Power off unused device PM resources
- ❑ Bonus: Extend usage of runtime PM

- Generic PM domains hook into runtime and system PM
- Extensible with dev_pm_genpd_set_performance_state()



Power On Device PM Resources at Probe ✅

- ✓ Work with existing DTs
- ✓ Remove SoC PM details from platform drivers
- ✓ Work with existing drivers that use PM resources directly
- ✓ Provide consistent device power states
- ✓ **Power on device PM resources at probe**
- ❑ Power off device PM resources at remove
- ❑ Bonus: Power off unused device PM resources
- ❑ Bonus: Extend usage of runtime PM

struct dev_pm_domain::activate()

struct generic_pm_domain::power_on()

TODO: Allow generic_pm_domains to come from software

Power On Device PM Resources at Probe ✅

- ✓ Work with existing DTs
- ✓ Remove SoC PM details from platform drivers
- ✓ Work with existing drivers that use PM resources directly
- ✓ Provide consistent device power states
- ✓ Power on device PM resources at probe
- ✓ **Power off device PM resources at remove**
- ❑ Bonus: Power off unused device PM resources
- ❑ Bonus: Extend usage of runtime PM

struct dev_pm_domain::dismiss()

struct generic_pm_domain::power_off()

TODO: Allow generic_pm_domains to come from software

Power Off Unused PM Resources ✅

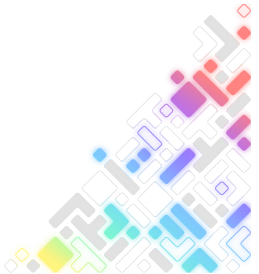
- ✓ Work with existing DTs
- ✓ Remove SoC PM details from platform drivers
- ✓ Work with existing drivers that use PM resources directly
- ✓ Provide consistent device power states
- ✓ Power on device PM resources at probe
- ✓ Power off device PM resources at remove
- ✓ **Bonus: Power off unused device PM resources**
- ❑ Bonus: Extend usage of runtime PM

- Avoid requiring drivers to be bound
- Use an SoC PM domain as a catch-all
  - Parent of other PM domains created in SoC driver
  - status = “disabled” devices are managed by this PM domain
- Power off PM domains during system suspend
  - Even if the associated device is unbound
- Leave device runtime PM state decision to userspace
- Enable runtime PM autosuspend for devices created in SoC driver

Extend usage of runtime PM ✅

- ✓ Work with existing DTs
- ✓ Remove SoC PM details from platform drivers
- ✓ Work with existing drivers that use PM resources directly
- ✓ Provide consistent device power states
- ✓ Power on device PM resources at probe
- ✓ Power off device PM resources at remove
- ✓ Bonus: Power off unused device PM resources
- ✓ **Bonus: Extend usage of runtime PM**

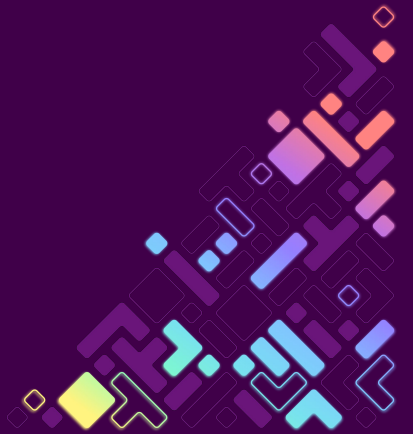
Usage of runtime PM mandatory to power on or off PM resources at runtime



Conclusion

- ✓ Work with existing DTs
- ✓ Remove SoC PM details from platform drivers
- ✓ Work with existing drivers that use PM resources directly
- ✓ Provide consistent device power states
- ✓ Power on device PM resources at probe
- ✓ Power off device PM resources at remove
- ✓ Bonus: Power off unused device PM resources
- ✓ Bonus: Extend usage of runtime PM

Questions?



References

1. https://lore.kernel.org/all/1418226513-14105-1-git-send-email-a.hajda@samsung.com/
2. https://static.sched.com/hosted_files/osseu18/0f/deferred_problem.pdf
3. aa42240ab254 ("PM / Domains: Add generic OF-based PM domain look-up")
4. f721889ff65a ("PM / Domains: Support for generic I/O PM domains (v8)")
5. e5cc8ef31267 ("ACPI / PM: Provide ACPI PM callback routines for subsystems")
6. https://learn.microsoft.com/en-us/windows-hardware/drivers/kernel/device-power-states
7. https://lore.kernel.org/all/1623682.7KVblAB3KQ@vostro.rjw.lan/
8. 9ed9895370ae ("driver core: Functional dependencies tracking support")
9. https://lore.kernel.org/all/561E1378.6000906@collabora.com/